Journal of Circuits, Systems, and Computers
Vol. 28, No. 12 (2019) 1950213 (21 pages)

DOI: 10.1142/S021812661950213X

World Scientific
www.worldscientific.com

A Group-Based Buffer Management for SSD*

Yan Liu†, Jilong Xu‡ and Guoqi Xie§

College of Computer Science and Electronic Engineering,
Hunan University, Changsha, Hunan, P. R. China
†*liuyan@hnu.edu.cn*
‡*xujl@hnu.edu.cn*
§*xgqman@hnu.edu.cn*

Zaimei Zhang

School of Economics and Management,
Changsha University of Science and Technology,
Changsha, Hunan, P. R. China
zhangzaimei@163.com

Keqin Li

Department of Computer Science,
State University of New York,
Albany, NY 12222, USA
lik@newpaltz.edu

Received 30 June 2018
Accepted 6 December 2018
Published 8 January 2019

Random writes limit the application of SSDs significantly because of their poor latency and high garbage collection overhead. Traditional page-based and block-based buffer management algorithms cannot achieve both high buffer hit ratio and good destage sequentiality at the same time. In this paper, we propose a hybrid scheme called the group-based buffer management (GBBM). To improve buffer hit ratio and decrease write/erase counts, GBBM divides buffer space into *Page Region* and *Group Region*. The frequently accessed data pages are placed at the *Page Region*, while infrequently accessed random written data are grouped in the *Group Region*. GBBM has been evaluated extensively through simulations. The write counts of GBBM show an average decrease of 12.7% compared with page-level buffer scheme. Compared with hybrid buffer management such as CBM, GBBM decreases the average write/erase count by 14.3%/12.1%. The write hit ratio of GBBM shows a 4.5% improvement as compared with PAB. The proposed GBBM can significantly reduce the number of write operations while maintaining a relatively high buffer hit ratio.

Keywords: Buffer management; cache; erase counts; write count.

*This paper was recommended by Regional Editor Tongquan Wei.
§Corresponding author.

1. Introduction

Solid-state disks (SSDs) have been revolutionizing the storage system during the past decades and have consequently gained strong interest in both data centers and high-performance computing.[1–5] Compared with traditional hard disk drives (HDDs), SSDs are built of semiconductor memory, with no moving part, and thus can provide low access latency, high data bandwidth, lower power consumption and has no noise.[6,7]

Although SSDs have a number of advantages over rotating disks, they suffer from the different read and write access speeds and the limitations of their lifespan particularly when applied in the enterprise environment.[8–10] The performance of an SSD is highly correlated with access patterns. Random writes are much slower than sequential writes in an SSD due to the properties of NAND flash memory internal mechanism. Because flash blocks wear out after repeated writes, SSDs have the potential to wear out. Data stored in flash memory is no longer reliable when the amount of erasure exceeds the maximum erase times. Thus, random write affects the performance and the lifespan of SSDs seriously.

To make full use of access patterns, numerous works have focused on the buffer management scheme design of SSDs. A good buffer management scheme should reduce the effects of random writes and suit access patterns. Generally, buffer management algorithms can be divided into three categories: page-based, block-based and hybrid, such as Clean-First Least Recently Used (CFLRU),[11] Popularity-Aware Buffer (PAB)[12] and Cooperative Buffer Management (CBM),[13] respectively. Their works contribute considerably to the performance of SSDs by organizing data access in the buffer. Compared with the page-based buffer management, the block-based scheme has fewer write/erase counts, but it sacrifices the buffer utilization of SSDs. The hybrid buffer management is a tradeoff between page-based and block-based methods.

In general, page-based buffer managements have high buffer hit ratio and high write/erase counts. Block-based buffer managements have low write/erase counts. They group resident pages in write buffer and flush them as sequentially as possible. However, because of the coarse management granularity and replacement policy, the hot data pages in the evicted block are also flushed to flash resulting in low buffer hit ratio. Block-based buffer managements merge clean pages into a victim block. The merge operation will cause large numbers of read operation and increase unnecessary data migration between buffer and flash memory. Hybrid buffer management such as CBM coordinates write buffer and read cache. CBM merges clean pages of read cache into the victim block using the merged-on-flush operation. This operation improves the destage sequentiality and decreases the response time of SSD. However, it has extra write operations when the block is flushed to flash. Buffer utilization and write/erase counts of SSDs are difficult to balance.

In this paper, we present a novel hybrid buffer management scheme named group-based buffer management (GBBM). GBBM divides buffer into the *Page* and *Group Regions*. Data access requests are disposed in the *Page Region* while the evicted block is sorted in the *Group Region*. The GBBM reserves pages with high temporal locality in the *Page Region* and groups the longest unused clean pages into the block in the *Group Region*. When a block is evicted from the *Group Region*, GBBM prioritizes the clean block (the block formed with clean pages) as the victim block. We compare our scheme with other common flash-aware cache algorithms using SSDSim and show that GBBM not only reduces block write counts to extend SSD lifespan, it also has the same level of buffer hit ratio with classical page-based buffer scheme.

We make the following contributions:

- We design a new hybrid buffer management scheme with a high buffer hit ratio through the full use of temporal and spatial localities in the *Page Region*. The scheme has low write counts by grouping the unused clean pages into victim block in the *Group Region* when the block is flushed into flash.
- An adjustable threshold is proposed to suit different workloads by dividing the buffer area into the *Page Region* and *Group Region* in different sizes.
- We implement GBBM on SSDSim. Our extensive experiment results show that GBBM can reduce the erase counts by 12.1% compared with the state-of-the-art buffer management schemes.

The rest of this paper is organized as follows. Section 2 provides an overview of background and related works. In Sec. 3, we present the design details of GBBM. Performance evaluations of GBBM are presented in Sec. 4. Finally, we conclude the paper in Sec. 5.

2. Background and Related Works

2.1. *SSD and buffer*

SSDs have different performance characteristics than HDDs. The unit of read and write operations of SSDs is a page, and a page can be written only after the entire block to which it belongs has been erased.[14] The basic design principle of an SSD buffer is to keep the most recently and frequently accessed data in the buffer as long as possible so as to effectively improve the performance of read/write operations and further reduce the response time of the requests. According to the characteristics of flash memory, read speed is significantly faster than write,[15] and SSD is not good at dealing with random writes. Based on these two facts, different buffer management algorithms have different effects on the performance of SSD. In this paper, we focus on improving write performance by a group-based buffer management design.

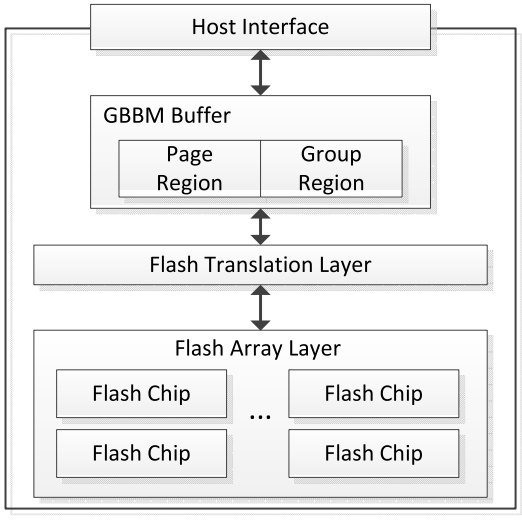


Fig. 1. Architecture of SSDs using GBBM.

Figure 1 shows the overview of proposed GBBM in architectural view of an SSD device. SSDs are usually composed of host interface, RAM buffer, Flash Translation Layer (FTL)[16] and Flash Array Layer (FAL).[17] The RAM buffer, acting as a data buffer for SSDs, can significantly improve the SSDs' performance and extend their life. SSDs' controller uses DRAM as the buffer to improve performance. Using the proposed GBBM, the RAM buffer of an SSD becomes a hybrid buffer with different granularities by dividing it into *Page Region* and *Group Region*. The FTL allows hosts to access the flash memory as conventional disk drives. It performs address mapping, garbage collection and wear leveling. According to the address mapping schemes, the FTL can be classified into three types: page-level, block-level and hybrid, such as BAST,[18] FAST,[19] LAST,[20] SAST,[21] DFTL[22] and NFTL.[23] A page-level FTL is used as the default FTL in this work because it has the best hit ratio.

In this paper, hybrid buffer management and universal feature servicing both read and write accesses are proposed first. Then a locality-aware replacement policy is designed to manage the data transmission between *Page* and *Group Regions* of GBBM.

2.2. *Random writes and aggregation*

Random writes are much slower than sequential writes. Performance of mixed workload with sequential and random writes is even more worse than the random writes. Figure 2 shows the write performance of Intel 600 PM.2 2280 SSD by using IOmeter in our experiment platform. For 4-kB request, sequential write speed is

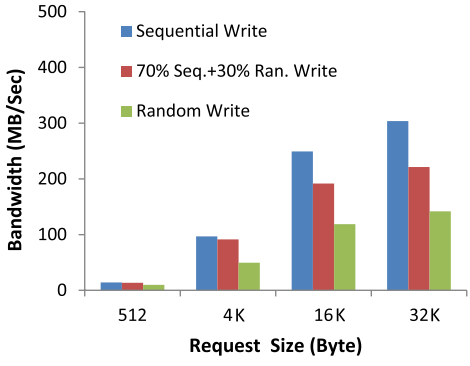


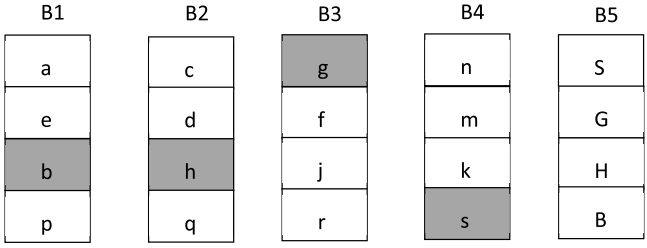
Fig. 2. Write performance of an SSD.

96-MB/s while random write is 49-MB/s. Write performance is decreased obviously when the sequential write request is mixed with random write. For 16-kB request and mixed workload (70% sequential and 30% random writes), the write speed is decreased by 23.3%. The result reflects that random writes limit the SSD performance significantly. The random writes have the following issues.

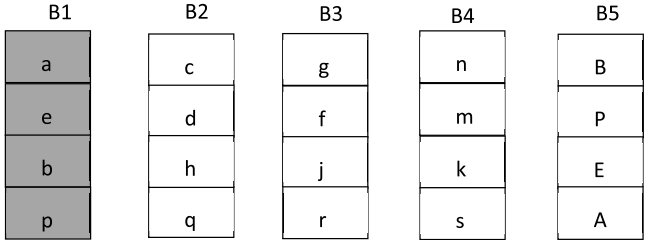
- SSDs' lifetime: Random writes increase erasure of flash memory, causing the SSDs to wear out much faster than sequential write[24] and thereby reducing their lifespan.
- Garbage collection overhead: Random writes result in greater garbage collection overhead compared to sequential writes. This is because random writes generate a larger number of invalid pages and trigger garbage collection operations more likely than sequential writes.
- Internal fragmentation: If incoming write operations are distributed randomly over the logical block address space, then the related physical flash memory blocks will be fragmented.[25]
- Internal parallelism: While striping and interleaving using advanced features of SSDs can improve sequential write performance, their ability to deal with random write is very limited.[26]

Therefore, random writes have many negative effects on SSDs. Data aggregation is one of the ways to improve SSDs' performance considering random writes. Aggregation is the basic idea of block-based cache management algorithms, i.e., the pages that belong to the same physical block in the cache are clustered together.[27]

Figure 3 shows an example of the aggregation operation between five physical blocks, namely, $B1$–$B5$. Each physical block consists of four physical pages, and the gray portion represents an invalid data page. $B5$ is the target block for storing updated data. In Fig. 3(a), S, G, H and B are the updated pages. We store them in $B5$



(a) General operation.



(b) Aggregation operation.

Fig. 3. General and aggregation operations.

and set the corresponding pages s, g, h and b to invalid, which is called the out-place update. Thus, each block has an invalid page at this time. If the garbage collection procedure occurs, $B1$–$B4$ are selected as the targets. The erase operation needs to copy back three pages of each block. In Fig. 3(b), B, P, E and A are the updated pages, the corresponding old pages are stored in $B1$. If the garbage collection operation is performed at this time, $B1$ will be selected as the target for garbage collection because all the four data pages in $B1$ are invalid. Therefore, the block erase operation can be performed without moving the internal pages.

We place data pages that belong to the same physical block together to reduce unnecessary internal data migration. When the SSDs' buffer is full, the aggregated data blocks are selected and removed from the buffer first using our page-based aggregation policy which increases the number of invalid data pages in the corresponding aggregated data blocks. When the SSD begins garbage collection, the possibility of the aggregated data blocks to be selected as target is increased, which can decrease the overhead of garbage collection operation.

2.3. *Buffer schemes*

Based on the different granularities of buffer management schemes, they can be divided into page-based, block-based and hybrid methods.

The page-based buffer management schemes organize and replace buffered data at page granularity, such as LRU, CFLRU,[11] FARS[28] and 2QW-Clock.[29] These

schemes attempt to increase the buffer hit ratio as much as possible. They focus on utilizing temporal and spatial localities to predict the next page to be accessed and minimize page fault ratio. However, these algorithms may increase the number of stored debris in the flash memory, which would significantly increase the cost of garbage collection.

CFLRU is a classical page-based buffer management scheme. CFLRU divides the buffer area into working and clean-first regions. It uses the characteristic of flash memory that the overhead of write operation is greater than read operation, and chooses a clean page as a victim rather than dirty pages when the buffer area is full. However, the size of the clean-first region in the CFLRU algorithm is determined by the size of the window. It is difficult to determine a suitable window size to suit different tasks. A large window (the clean-first region) will decrease the hit ratio, a small window will increase the overhead of flush. Moreover, CFLRU does not optimize the sequentiality of evicted pages, thus, many random writes will have negative impacts on SSDs' performance.

Flash-aware buffer management (FAB),[27] BPLRU,[30] CFDC[31] and PAB are commonly used block-based buffer management schemes. Resident pages in the buffer are grouped on the basis of their logical block associations. Thus, the block-based scheme decreases the number of stored debris and the cost of garbage collection. Nevertheless, according to these algorithms, accessing a logical page results in adjusting all pages in the same logical block having the same recency. FAB is a classical block-based buffer management scheme. It buffers the data in the form of data block to organize an LRU-linked list. FAB merges data pages belonging to the same block, thus can reduce flash writes and erase operations by preferentially replacing data blocks which have the largest number of data pages. But the performance of FAB decreases when the requests are random. In addition, FAB selects the data block with the most data pages as the victim block. If the buffer has many blocks with higher access frequency but fewer data pages, FAB cannot adapt to this situation very well, thus decreasing the hit ratio and the overall I/O performance.

Hybrid managements, such as CBM, are proposed to combine both page-based and block-based buffer management methods' advantages for SSDs. CBM coordinates write buffer and read cache to improve SSDs' performance. The write buffer is divided into two regions, a page region and a block region. When the number of pages belonging to the same block reaches a threshold in page region, these pages will be migrated to the block region. When the buffer is full, CBM preferentially selects a block in the block region as a victim block and merges the dirty pages in victim block. During the merge operation, extra read and write operations must be invoked in addition to the necessary erase operation due to the copying of valid pages of the data block. Therefore, merge operation degrades the performance of SSDs because of the extra data movement. And the cache proposed by CBM has two parts, one is read-only cache and the other is write-only cache. It uses two different physical memory devices. Physical costs are high compared to GBBM. At the same cache size,

the CBM is physically partitioned into a read cache and a write cache. Therefore, CBM cannot flexibly adapt to various application scenarios.

All of these buffer management schemes of SSDs focus mainly on how to improve the buffer hit ratio, however, full attention has not been given to the effect of write-amplification and write/erase counts. This work reduces the write/erase counts to extend SSDs' lifespan and obtains the same level of buffer hit ratio as other page-based buffer algorithms by using hybrid buffer and aggregation operation.

3. Design of GBBM

In this section, we first introduce the basic idea of our proposed GBBM scheme. Then we describe the buffer management policy used in GBBM. Finally, we present the brief description of how to choose the threshold of partition.

3.1. *Basic idea of GBBM*

The proposed GBBM aims to significantly reduce the number of write operations while maintaining a relatively high buffer hit ratio.

Figure 4 shows the basic idea of GBBM. Compared with existing hybrid buffer algorithm CBM which coordinates write buffer and read cache to improve SSDs' performance, GBBM processes the read and write requests simultaneously. GBBM divides the buffer into the *Page* and *Group Regions*. In the *Page Region*, GBBM uses page-based algorithm such as LRU to order pages and decides which page will be evicted. In the *Group Region*, GBBM uses group operation to aggregate pages evicted from *Page Region* into the *Group Node*, which is the basic unit of *Group Region*. GBBM has high buffer hit ratio because of the page-based management algorithm used in *Page Region* and low write/erase counts by grouping the victim pages into *Group Node* in *Group Region*. An adjustable threshold is proposed to divide the buffer into *Page Region* and *Group Region* to suit different workloads.

Hence, GBBM can obtain high buffer hit ratio in *Page Region* and decrease the erase counts when the data block is flushed to flash memory in *Group Region*.

3.2. *Buffer management*

3.2.1. *Hybrid buffer management*

The SSDs' buffer has strict limitation of capacity and cost. The buffer management scheme should maintain high hit ratio and decrease the write/erase counts to take full advantage of this small area. GBBM divides the SSDs' buffer into two areas: *Page Region* and *Group Region* as shown in Fig. 4. The read or write requests can be served in the *Page Region* directly if they are hit. When they are miss in the *Page Region*, we will process these requests in the *Group Region* or flash memory. When a page is hit in the *Group Region*, it will be moved to the *Page Region* to take advantage of locality.

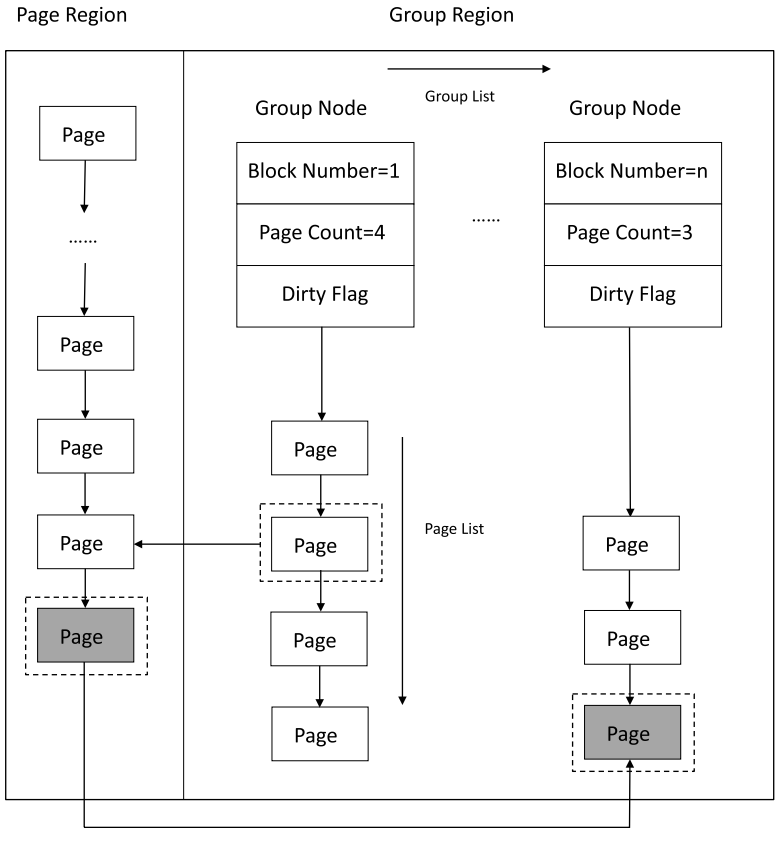


Fig. 4. Management of GBBM.

In Fig. 4, *Group Node* is the basic unit in *Group Region* that includes *Block Number*, *Page Count* and *Dirty Flag*. *Block Number* represents a physical block number (PBN). All of the data pages in the physical block are managed by a *Page List*. *Page Count* stands for the number of data pages in this physical block. *Dirty Flag* indicates whether the *Page List* has dirty pages. The blocks in *Group Region* are organized as *Group List*, which is sorted by *Page Count* (see in Fig. 5). *Page Count* is the primary criterion to decide the position of a block in the *Block List*. The block that has maximum quantity of pages stays at the head of the *Group List*, and the *Group List* is sorted in descending order by *Page Count*. Blocks with dirty pages will remain in the *Group List*, and GBBM preferentially selects data blocks without dirty pages as victim blocks.

We use different replacement strategies in *Page Region* and *Group Region*. The pages in the *Page Region* are organized as page-based LRU list. For incoming read or write requests, they will be processed in the *Page Region* first. Data migration between *Page* and *Group Regions* is a core module in GBBM.

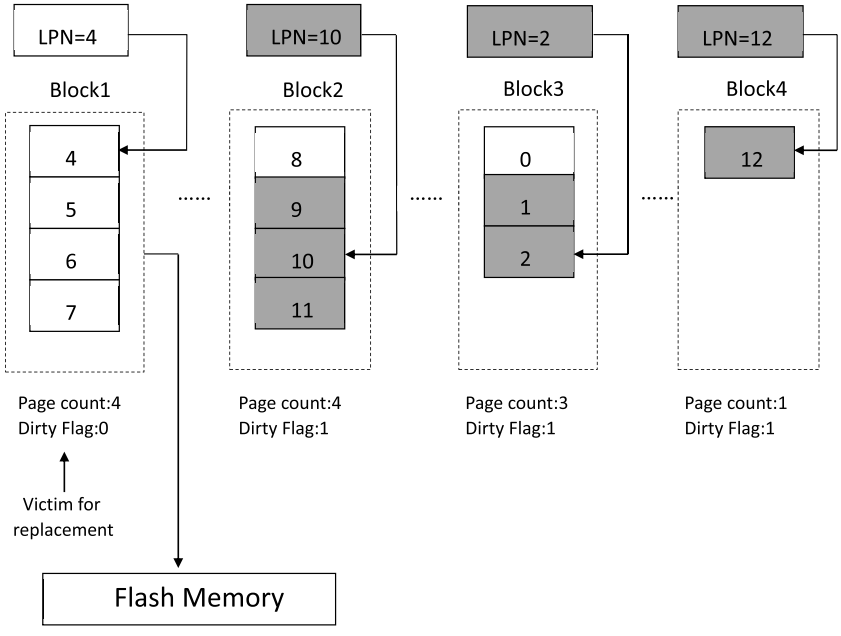


Fig. 5. An example of flush.

Algorithm 1 shows the data management algorithm between the *Page* and *Group Regions*, namely migration between regions (MBRs). Read or write requests can be served in the *Page Region* directly if they are hit (lines 3 and 14). When the accesses are miss in the *Page Region*, we will process them in the *Group Region* (lines 5, 6, 16 and 17) or flash memory. When a page is hit in the *Group Region* or flash memory, it will be migrated to the *Page Region* to take advantage of locality (lines 6 and 9).

If the *Page Region* is full, page that was last accessed (the last one of the *LRU list* in this paper) is evicted to the *Group Region*. We calculate the PBN according to the logical page number (LPN) of data page and determine whether the corresponding physical block node exists in the *Group List*. For example, in Fig. 4, the gray page in the *Page Region* will be evicted and its PBN is 2. Then we put this data page into the *Page List* that belongs to the *Group Node* which has *Block Number* = 2. Furthermore, GBBM migrates the page from the *Group Region* to *Page Region* if the page is hit. Thus, the locality of data page is used efficiently by the flexible movement between the *Page* and *Group Regions*.

3.2.2. *Flush policy*

When the SSDs' buffer is full, replacement operation is triggered by flush policy to produce more space using Algorithm 2, namely, group region management (GRM). The flush operation creates many data movements between the buffer and SSDs'

Algorithm 1. MBRs: Migration Between Regions

Require: Request R with logical page number LPN, request size SIZE, request type TYPE

```
1: if (TYPE == READ) then
2:     if (LPN in Page Region) then
3:         Read data from Page Region;
4:     else if (LPN in Group Region) then
5:         Read data from Group Region;
6:         Copy data to Page Region;
7:     else
8:         Read data from the flash to respond to the request;
9:         Copy data to Page Region;
10:    end if
11: end if
12: if (TYPE == WRITE) then
13:    if (LPN in Page Region) then
14:        Update the data;
15:    else if (LPN in Group Region) then
16:        Remove the page from Group Region to Page Region;
17:        Update the data in Page Region;
18:    end if
19: end if
20: if (Page Region is full) then
21:    Select the victim page X;
22:    Call Group_Region_buffer_management(page X);
23:    Store the new request in Page Region;
24: end if
```

flash array. A different flush policy will produce a different request sequence, and the request sequence can directly affect the performance and lifespan of SSD. The cost of random writes miss is much higher than that of sequential writes. We use GRM to suit access patterns and extend the lifespan of SSDs by keeping popular data in buffer as long as possible. The random write pages evicted from *Page Region* are grouped into sequential writes by the *Group Node* in the *Group Region*.

In Algorithm 2 (GRM), when the *Group Region* is full, GBBM chooses the block containing maximum data pages to write to flash memory (line 2). If more than one *Group Node* has the same number of data pages, the *Group Node* with the largest number of clean pages is selected as a victim block (line 5). If the victim group has dirty pages, these dirty and clean pages are flushed into flash together (line 8). GRM creates a new *Group Node* if the *Group Node* list does not contain a node with the same block number as the page migrated from *Page Region* (line 15).

Algorithm 2. GRM: Group Region Management

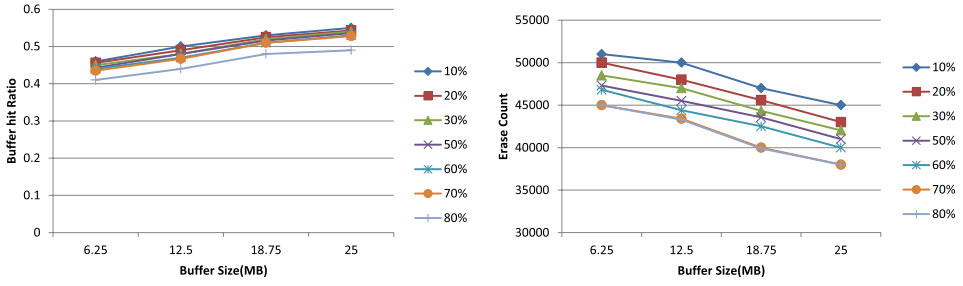
Require: Page X, the block number (Bln) of page X;

if (*Group Region* is full) **then**
    Select the *Group Node* whose *Page Count* is maximum as the victim group;
end if
if (multiple *Group Nodes* have same pages) **then**
    Select the *Group Node* with the most clean pages;
end if
if (there are dirty pages in the victim group) **then**
    Flush both dirty and clean pages of the victim group;
else
    Discard all the pages of the victim group;
end if
if (there is a *Group Node* in the *Group List* whose *Block Number* == Bln) **then**
    Insert the page X in the *Page List* of the selected *Group Node*;
else
    Create a *Group Node*, then insert the page X into it;
end if

A *Group Node*, which is a combination of dirty pages and clean pages, is sequentially written into the flash array once the *Group Node* is selected as a victim block. Hence, our replacement policy chooses the *Group Node* with the most clean pages as the victim block to be flushed. If the *Group Node* does not contain dirty page, it need not be written to the flash array to update data. We can delay the progress of dirty pages being written to flash and decrease the write counts of flash array by using this strategy. Figure 5 shows a simple example. The white rectangle represents the clean page and the gray one is the dirty page. The *Page Count* indicates the number of pages in a *Group Node*. If a *Group Node* has a dirty page, the *Dirty Flag* is set to 1. When a replacement operation is required, the *Group Node* with the largest number of page count should be selected as the replacement target. In Fig. 5, the first and second *Group Nodes* contain four data pages. Because the first *Group Node* contains more clean pages, it is chosen as a replacement target. Thus, the dirty pages stay in the cache area as long as possible to improve access locality. The group operation make large numbers of random writes grouped into sequential writes, and GBBM can decrease the number of flash memory write operations and overhead of the garbage collection.

3.3. *Threshold of partition*

The *Page Region* is used to hold data pages that are often accessed and the *Group Region* is used to group the least used clean pages evicted from the *Page Region*.



(a) Buffer hit ratio. (b) Erase count.

Fig. 6. Evaluated results for different buffer sizes.

GBBM's objective is to decrease the amount of debris stored in the flash memory in the meantime to maintain high buffer hit ratio. The different division ratios of the *Group Region* and the *Page Region* can affect the buffer hit ratio and garbage collection efficiency.

To demonstrate how the threshold value affects the efficiency of the proposed GBBM, we test GBBM with *Financial*1 workload in different buffer sizes. We set the buffer sizes to 6.25, 12.5, 18.75 and 25 MB. The sizes of the *Group Region* are 10%, 20%, 40%, 50%, 60%, 70% and 80% of the buffer area. Figure 6 shows the evaluated results for different buffer sizes using the workload *Financial*1. We can see that the buffer hit ratio improved with the increase in buffer size. The buffer hit ratio stays at a relatively high level when the threshold is set to 70%. On the other hand, the erase count is decreased with the increase in buffer size because more data can be cached. The minimum erase counts for different buffer sizes can be obseved when the threshold is 70%. We set the partition threshold of the *Page* and *Group Regions* to 70% in our experiments. The partition threshold used here is suitable for this paper's workloads. Details on how to obtain the threshold are out of the scope of this paper.

4. Performance Evaluation

In this section, the performance of proposed GBBM is evaluated through extensive experiments.

4.1. *Experimental setup*

We implemented GBBM in SSDSim.[32] *Financial*1 and *Financial*2 are I/O traces from OLTP applications running at two large financial institutions.[33] We employ two traces provided by Microsoft and SNIA,[34] namely the *Radius* and *Radius-SQL*. Table 1 shows the attributes of the four workloads. *Financial*1 contains a large number of write requests. The average length of the write request is 7.5. *Financial*2 contains a large number of read requests. The average length of the read request is

Table 1. The characteristics of traces.

Trace	Read request	Write request	Average length of the read request	Average length of the write request
Financial1	1,235,596	4,099,351	4.5	7.5
Financial2	3,045,784	653,079	4.6	5.8
Radius	12,760	109,208	13.1	15.4
Radius-SQL	64,820	315,426	268.5	24.9

4.6. Radius is a small trace, but the ratio of read requests to write requests is 11.68%. The proportion of write requests is very large. Radius-SQL's read requests are less than write requests, but their average length is 268.5.

In this paper, four metrics are used to evaluate the behavior of different buffer management schemes including *buffer hit ratio*, *write count*, *write hit ratio* and *erase count*.

4.2. *Experimental results and discussion*

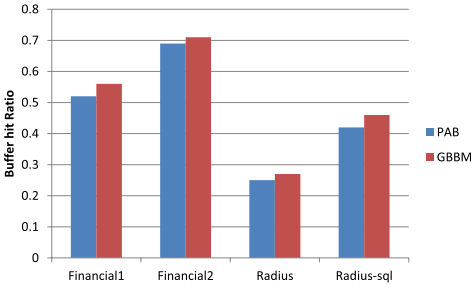
We compare GBBM algorithm with LRU, CFLRU, PAB and CBM. Figures 7–11 show the experimental results using workloads *Financial*1, *Financial*2, *Radius* and *Radius-SQL*.

4.2.1. *Buffer hit ratio*

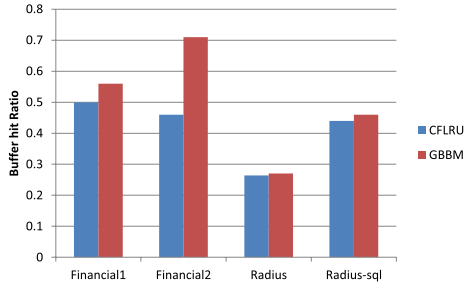
Figure 7 shows the buffer hit ratios of GBBM compared with other four algorithms. The design goal of PAB is to improve cache hit ratio and write sequencing. However, because PAB is a block-based algorithm, the buffer hit ratio of PAB is smaller than that of GBBM as shown in Fig. 7(a).

CFLRU is a classical page-based buffer management algorithm. In Fig. 7(b), GBBM obtains about 24% improvement compared with CFLRU using *Finacial*2 because it has a number of read requests and CFLRU deals with the clean pages first. Considering that the *Radius-SQL* has a large number of big requests and GBBM's group operation does not benefit from it, GBBM's buffer hit ratio is a little higher than CFLRU. In this experiment, GBBM can obtain better performance in comparison with CFLRU.

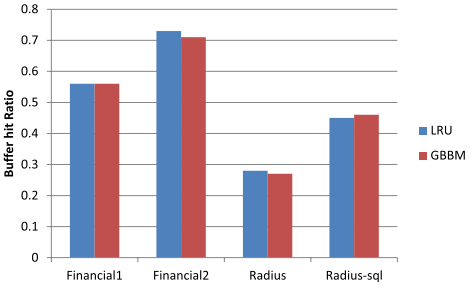
In Fig. 7(c), LRU's buffer hit ratio is only 0.2%, and 0.1% higher than GBBM using *Financial*2 and *Radius*. GBBM has the same buffer hit ratio as LRU using *Financial*1. Because GBBM first uses the *Page Region* through the LRU policy to serve access requests, thus its buffer hit ratio is very close to that of LRU. The buffer hit ratio of LRU is high because it only considers the temporal locality. However, the costs of read and write operations are different in SSDs, and the replacement cost of dirty pages is larger than that of clean pages. Though LRU has a little higher cache hit ratio compared with GBBM, but GBBM can distinguish dirty/clean pages from



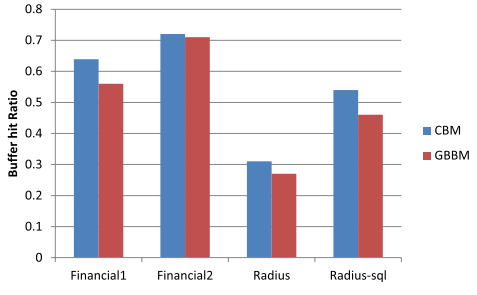
(a) Buffer hit ratio (PAB versus GBBM).



(b) Buffer hit ratio (CFLRU versus GBBM).



(c) Buffer hit ratio (LRU versus GBBM).



(d) Buffer hit ratio (CBM versus GBBM).

Fig. 7. Buffer hit ratios of GBBM compared with other four algorithms.

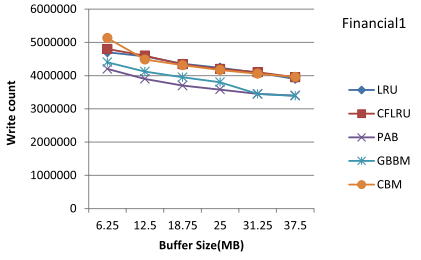
the access patterns. Thus, GBBM can decrease the number of erase counts and unnecessary write operations.

Figure 7(d) shows that the buffer hit ratio of CBM is the best because it coordinates the read cache in DRAM and write buffer in NVM. Using different buffers (DRAM and NVM) for read and write requests will significantly increase the cost of device and control algorithm. However, GBBM focuses on maintaining high cache hit ratio while significantly reducing the numbers of write and erase counts (we show the experimental data in the following paragraphs).

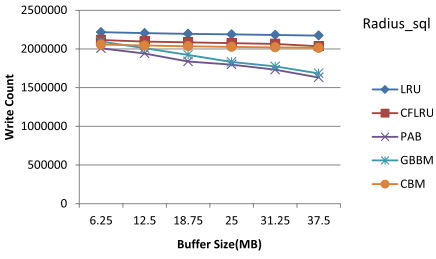
In general, GBBM can obtain a good buffer hit ratio compared with the current main algorithms.

4.2.2. *Write count*

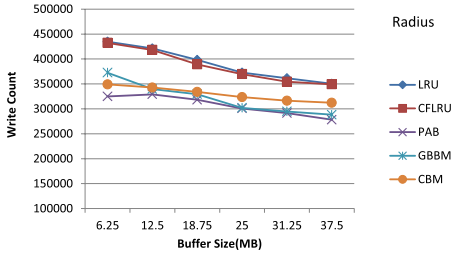
The less write counts a flash memory undergoes, the more longer is its life. Figure 8 shows the write counts of GBBM compared with other four algorithms using four workloads with increasing buffer size from 6.25 MB to 37.5 MB. With buffer size being increased, the write count of GBBM becomes closer to that of PAB, which is the block-based buffer algorithm. When the buffer size is 37.5 MB, the write count of GBBM is the same as that of PAB, and far less than the page-based buffers such as



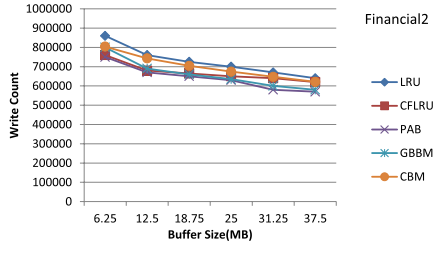
(a) Write count (*Financial* 1).



(b) Write count (*Radius-SQL*.)

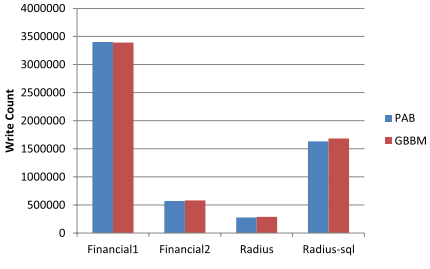


(c) Write count (*Radius*).

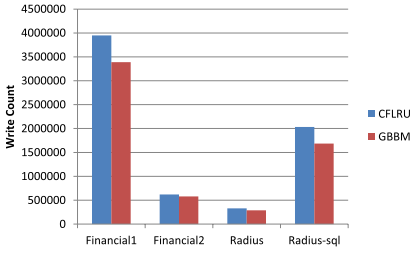


(d) Write count (*Financial* 2).

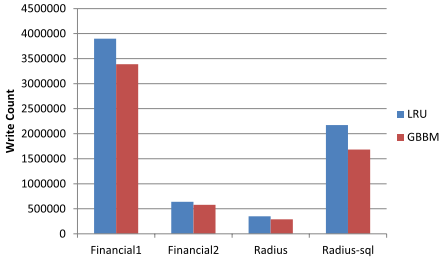
Fig. 8. Write counts at different buffer sizes.



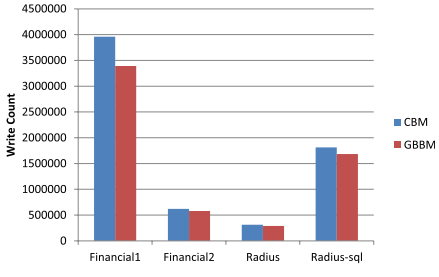
(a) Write count (PAB versus GBBM).



(b) Write count (CFLRU versus GBBM).

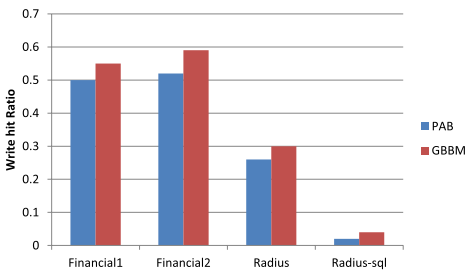


(c) Write count (LRU versus GBBM).

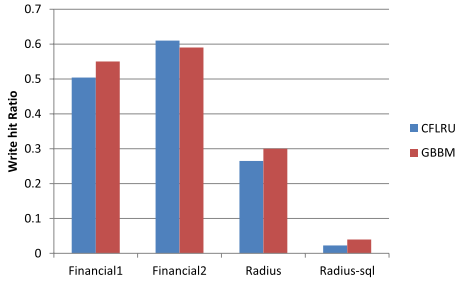


(d) Write count (CBM versus GBBM).

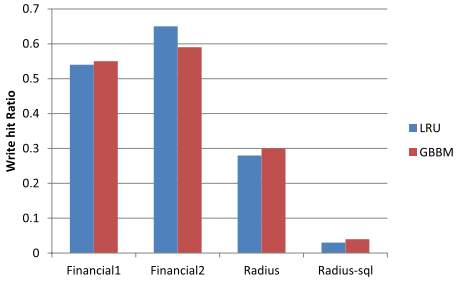
Fig. 9. Write counts when the buffer size is 37.5 MB.



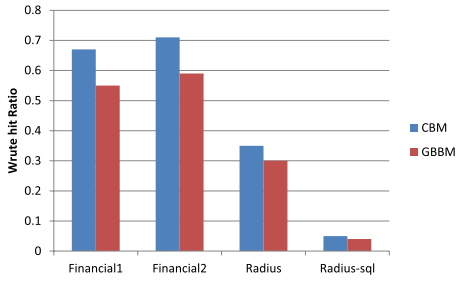
(a) Write hit ratio (PAB versus GBBM).



(b) Write hit ratio (CFLRU versus GBBM).



(c) Write hit ratio (LRU versus GBBM).



(d) Write hit ratio (CBM versus GBBM).

Fig. 10. Write hit ratios when the buffer size is 37.5 MB.

LRU and CFLRU. When the buffer size is 37.5 MB, the write counts of the GBBM compared with LRU and CFLRU are shown in Fig. 9.

Figure 9(a) shows the write count of GBBM compared with PAB using four workloads. We can see that GBBM has the same performance as PAB. As displayed in Figs. 9(b)–9(d), GBBM can decrease the write count by 10.8%, 13% and 16.7%, respectively, than CFLRU, LRU and CBM using *Financial*1 because it has a large number of small write requests. Compared with page-based buffer management schemes (such as CFLRU and LRU), GBBM can group the clean pages and make the dirty pages stay in the buffer as long as possible. CBM merges the clean pages with the dirty pages when the block is flushed into the flash. Thus, CBM has many extra write operations compared with GBBM especially when the workload has large number of write operations. Figures 8 and 9 show that GBBM is a promising method to decrease the write count of SSDs.

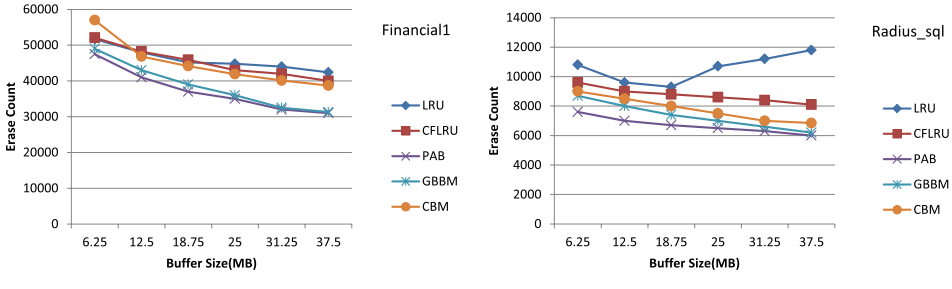
4.2.3. *Write hit ratio*

The overhead of write operation is greater than the read operation in SSDs. Increasing the write hit ratio of SSDs' buffer can yield considerable improvement in the performance of SSDs.

Then we evaluated the write hit ratios with different workloads using GBBM and other four algorithms. The buffer size is set to 37.5 MB. From Fig. 10(a), we can observe that the write hit ratio of GBBM is significantly greater than PAB. As shown in Figs. 10(b) and 10(c), the write buffer hit ratios are smaller than those of CFLRU and LRU algorithms using *Financial*2 because it contains a large percentage of read operations. Both read and write requests are buffered in GBBM. The read requests of *Financial*2 occupy most of the buffer area and GBBM does not have sufficient buffer area for the write requests. From Fig. 10(d), GBBM's write hit ratio does not show advantage compared with CBM in the case of containing large number of read requests. CBM responds to read and write requests in DRAM and NVM. This extra overhead of hardware used in CBM can obtain a high write hit ratio. In most cases, the write hit ratio of GBBM is better than PAB, CFLRU and LRU. These results show that GBBM obtains a better write hit performance with a relatively low cost and simple buffer management algorithm.

4.2.4. *Erase count*

Erase count has a serious effect on the garbage collection and service life of SSDs. *Financial*2 and *Radius* include few write operations, thus, the erase count of SSD is 0. We choose *Financial*1 and *Radius-SQL* as the workloads to evaluate the erase count performance. Figure 11 shows that the erase counts of GBBM are significantly less than the page-based buffer management algorithms such as LRU and CFLRU, and it very close to the block-based scheme PAB. The erase count of GBBM is also less than CBM, because CBM causes the extra write operations when flushing to flash. With the buffer size being increased, the erase counts of GBBM and PAB are more and more close. When buffer size is 37.5 MB, GBBM's erase count is the same as PAB and better than CBM. In other words, compared with page-based buffer schemes, such as LRU and CFLRU, GBBM can reduce the erase counts of flash memory more obviously, thereby extending the life of SSDs.



(a) *Financial*1. (b) *Radius-SQL*.

Fig. 11. Erase counts.

All the experiment results have confirmed that the proposed GBBM is an effective method to improve the performance of SSD and significantly reduce the number of write operations while maintaining a relatively high buffer hit ratio. GBBM helps firmware of SSD to provide high-performance I/O service to meet the growing data demands of applications.

5. Conclusion

In this paper, we present a hybrid buffer management scheme, GBBM, that minimizes write operations by exploiting both spatial and temporal localities. To improve both buffer hit ratio and destage sequentiality, GBBM divides the buffer area into *Page* and *Group Regions*. GBBM chooses the last page of LRU list in the *Page Region* to replace, and aggregates pages into a block in the *Group Region*. Because of the aggregation operation, GBBM can transform random write requests into sequential write requests and decrease the write count of flash memory. When a block is evicted from the write buffer, GBBM chooses the block which has the most clean pages as the target. Thus, GBBM can hold dirty pages in cache as long as possible. The experimental results demonstrate that the proposed GBBM can increase the cache hit ratio to the level of page-based buffer algorithms, but reduces the numbers of write and erase counts to the level of block-based buffer algorithms at the same time. In the future, we plan to propose more intelligent buffer management schemes for applications with complex access patterns and test them on a real hardware platform.

Acknowledgments

The authors would like to express their gratitude to the Editor and anonymous reviewers for their constructive comments, which have helped to improve the quality of the paper. This work was partially supported by the National Natural Science Foundation of China Under Grant Nos. 61872135 and 61702053, the Natural Science Foundation of Hunan Province (No. 2018JJ2066) and the Open Fund Project for Innovation Platform of Universities in Hunan (No. 11K002).

References

1. C. Min, K. Kim, H. Cho, S. W. Lee and Y. I. Eom, SFS: Random write considered harmful in solid state drives, *Proc. 10th USENIX Conf. File and Storage Technologies* (2012), p. 12.
2. R. S. Liu, C. L. Yang and W. Wu, Optimizing NAND flash-based SSDS via retention relaxation, *Proc. 10th USENIX Conf. File and Storage Technologies* (2012), p. 11.
3. Q. Yang and J. Ren, I-CASH: Intelligently coupled array of SSD and HDD, *Proc. 2011 17th IEEE Int. Symp. High Performance Computer Architecture* (2011), pp. 278–289.

4. A. Badam and V. S. Pai, SSDAlloc: Hybrid SSD/RAM memory management made easy, *Proc. 8th USENIX Conf. Networked Systems Design and Implementation* (2011), p. 16.
5. L. M. Grupp, J. D. Davis and S. Swanson, The bleak future of NAND flash memory, *Proc. 10th USENIX Conf. File and Storage Technologies* (2012), p. 2.
6. F. Douglis, R. Cáceres, M. F. Kaashoek, P. Krishnan, K. Li, B. Marsh and J. Tauber, Storage alternatives for mobile computers, *Proc. 1st USENIX Conf. Operating Systems Design and Implementation* (1994), p. 3.
7. R. Chen and M. Lin, Energy-aware buffer management scheme for NAND and flash-based consumer electronics, *IEEE Trans. Consum. Electron.* **61** (2015) 484–490.
8. S. Ahn, S. Hyun, T. Kim and H. Bahn, A compressed file system manager for flash memory based consumer electronics devices, *IEEE Trans. Consum. Electron.* **59** (2013) 544–549.
9. D. Lee, J. C. Kim, C. G. Lee and K. Kim, MRT-PLRU: A general framework for real-time multitask executions on NAND flash memory, *IEEE Trans. Comput.* **62** (2013) 758–771.
10. B.-K. Kim, S.-W. Lee and D.-H. Lee, h-Hash: A hash index structure for flash-based solid state drives, *J. Circuits Syst. Comput.* **24** (2015) 1550128.
11. S. Y. Park, D. Jung, J. U. Kang, J. S. Kim and J. Lee, CFLRU: A replacement algorithm for flash memory, *Proc. 2006 Int. Conf. Compilers, Architecture and Synthesis for Embedded Systems* (2006), pp. 234–241.
12. X. Guo, J. Tan and Y. Wang, PAB: Parallelism-aware buffer management scheme for NAND-based SSDS, *Proc. 2013 IEEE Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2014), pp. 101–110.
13. Q. Wei, C. Chen and J. Yang, CBM: A cooperative buffer management for SSD, *Proc. 2014 30th Symp. Mass Storage Systems and Technologies* (2014), pp. 1–12.
14. T.-S. Chung, D. J. Park and J. Kim, An efficient flash translation layer for large block NAND flash devices, *J. Circuits Syst. Comput.* **24** (2015) 1550138.
15. D. B. Yeo, J.-Y. Paik and T.-S. Chung, Hierarchical request-size-aware flash translation layer based on page-level mapping, *J. Circuits Syst. Comput.* (2018), doi: 10.1142/SO218126619501172.
16. S. G. Ayele, R. Jin, S. J. Kwon, M. Attique and T.-S. Chung, Efficient FTL-aware data categorization and identification scheme for flash memory, *J. Circuits Syst. Comput.* **24** (2015) 1550113.
17. D. Ma, J. Feng and G. Li, A survey of address translation technologies for flash memories, *ACM Comput. Surv.* **46** (2014) 1–39.
18. J. Kim, J. M. Kim, S. H. Noh, S. L. Min and Y. Cho, A space-efficient flash translation layer for compactflash systems, *IEEE Trans. Consum. Electron.* **48** (2002) 366–375.
19. S. W. Lee, D. J. Park, T. S. Chung, D. H. Lee, S. Park and H. J. Song, A log buffer-based flash translation layer using fully-associative sector translation, *ACM Trans. Embed. Comput. Syst.* **6** (2007) 18.
20. S. Lee, D. Shin, Y. J. Kim and J. Kim, LAST: Locality-aware sector translation for NAND flash memory-based storage systems, *ACM SIGOPS Oper. Syst. Rev.* **42** (2008) 36–42.
21. J. U. Kang, H. Jo, J. S. Kim and J. Lee, A superblock-based flash translation layer for NAND flash memory, *Proc. 6th ACM & IEEE Int. Conf. Embedded Software* (2006), pp. 161–170.
22. A. Gupta, Y. Kim and B. Urgaonkar, DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings, *ACM SIGARCH Comput. Archit. News* **37** (2009) 229–240.

23. Z. Liu, L. Yue, P. Wei, P. Jin and X. Xiang, An adaptive block-set based management for large-scale flash memory, *Proc. 2009 ACM Symp. Appl. Comput.* (2009), pp. 1621–1625.
24. H. Kim and U. Ramachandran, Flashlite: A user-level library to enhance durability of SSD for P2P file sharing, *Proc. 2009 29th IEEE Int. Conf. Distributed Computing Systems* (2009), pp. 534–541.
25. F. Chen, D. A. Koufaty and X. Zhang, Understanding intrinsic characteristics and system implications of flash memory based solid state drives, *ACM SIGMETRICS Perform. Eval. Rev.* **37** (2009) 181–192.
26. A. Rajimwale, V. Prabhakaran and J. D. Davis, Block management in solid-state devices, *Proc. 2009 Conf. USENIX Annu. Technical Conf.* (2009), p. 21.
27. H. Jo, J. U. Kang, S. Y. Park, J. S. Kim and J. Lee, FAB: Flash-aware buffer management policy for portable media players, *IEEE Trans. Consum. Electron.* **52** (2006) 485–493.
28. O. Kwon, H. Bahn and K. Koh, FARS: A page replacement algorithm for NAND flash memory based embedded systems, *Proc. 2008 8th IEEE Int. Conf. Computer and Information Technology* (2008), pp. 218–223.
29. D. He, F. Wang, D. Feng, J. N. Liu, Y. X. Wu, Y. Hu and Y. He, 2QW-CLOCK: An efficient SSD buffer management algorithm, *Proc. 2015 IEEE 22nd Int. Conf. High Performance Computing* (2016), pp. 47–53.
30. H. Kim and S. Ahn, BPLRU: A buffer management scheme for improving random writes in flash storage, *Proc. 6th USENIX Conf. File and Storage Technologies* (2008), pp. 239–252.
31. Y. Ou, T. Härder and P. Jin, CFDC: A flash-aware buffer management algorithm for database systems, *ADBIS 2010*: *Advances in Databases and Information Systems*, Lecture Notes in Computer Science, Vol. 6295 (Springer, Berlin, 2010), pp. 435–449.
32. Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo and S. Zhang, Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity, *Proc. Int. Conf. Supercomputing* (2011), pp. 96–107.
33. Laboratory for Advanced System Software, UMASS Trace Repository; OLTP Application I/O (2015), http://traces.cs.umass.edu/index.php/storage/storage.
34. SNIA IOTTA Repository, Block I/O Traces (2016), http://iotta.snia.org/tracetypes/3.